

AN INTERACTIVE 3D WORLD BUILT ON A TRANSACTIONAL OPERATING SYSTEM

M. Fakler, S. Frenz, R. Göckelmann, M. Schöttner, P. Schulthess
Department of Distributed Systems, University of Ulm, Germany
fakler@vs.informatik.uni-ulm.de

Abstract

Multi-player games and interactive 3D worlds are a fast growing market. Traditional server-based implementations struggle with distribution and synchronisation of objects. Typically, sophisticated middleware is used to distribute objects but consistency remains the responsibility of the programmer. Sharing objects within a shared memory is an alternative offering transparent distribution and consistency for all objects. Such a facility is offered by the Plurix operating system developed for PC clusters in which consistency and synchronisation of the shared objects is guaranteed by a novel transaction based consistency model.

In this paper we describe the design of our scene graph and an efficient multistage rendering strategy for the visualisation process of an interactive 3D world inside the Plurix system. Measurements demonstrate the advantages of building an interactive world on top of a transactional shared memory system.

Keywords: *Virtual Reality & Visualisation, Transactional Consistency, Computer Networks & Systems*

1. INTRODUCTION

Modern multi-player games, virtual reality chat systems, and interactive 3D online worlds exhibit an increase in acceptance, popularity and distribution of these applications. Traditionally, they are implemented on client-server architectures or single machines with the help of sophisticated middleware for communication with other nodes. But the programmer still has to solve distribution issues and to sustain consistency and synchronisation in his application. Weak consistency models may be used to speed up the performance of the developed application but this increases the complexity for the programmer even more.

An alternative is to use a distributed operating system (OS). The Plurix OS offers transactional consistency and strong synchronisation for all objects in the cluster. Thus the programmer can focus completely on the development of the desired application without the need of solving additional tasks concerned with the distribution of data.

Although shared memory systems are well known in the scientific world they have only been used for special number crunching applications. We find that a shared memory model is also suitable for other types of distributed applications especially those with high communication volume between nodes like an interactive virtual reality environment.

Therefore we have built an 3D world on top of the transactional Plurix OS to demonstrate how its benefits may be used to implement those client-server based applications in an easy and efficient manner within a distributed OS architecture. We have designed a common scene graph adapted to a shared memory environment and an associated transactional render engine for the visualisation process.

In the first section of this paper we present a short overview of the Plurix OS followed by a section of related work. Subsequently, we describe our approach of the shared scene graph and the implemented visualisation process. Finally, we discuss the measured performance figures and give our conclusions.

2. PLURIX – A TRANSACTION BASED OPERATING SYSTEM

The Plurix project [1] implements a lean operating system for PC clusters. It works in a fully object-oriented way and is written almost entirely in Java. The Plurix system itself is compiled with our own Plurix Java Compiler (PJC) directly into native Intel machine code.

2.1 Memory Model

Plurix uses the concept of a distributed shared memory (DSM) instead of message passing or remote method invocation to communicate with other nodes of the cluster. All nodes have the same view of the objects in the DSM which creates a single-system-image (SSI) perspective for the programmer. Publishing objects within the DSM is easily achieved via the global name-service. The distribution of the objects to other nodes during an access is done completely transparent by the operating system.

Persistence in Plurix is offered by an optional page-server taking incremental checkpoints and accomplishing automatic fallbacks in case of a failure.

2.2 Consistency Model

Plurix introduces a novel consistency model named transactional consistency [2]. All calculation executed by a node in the cluster is encapsulated within a transaction (TA) much like in a traditional database system. When a TA finishes it is validated against all stations in the cluster. As a consequence accesses to a shared object always return the newest version of the corresponding data.

If a TA writes an object which is simultaneously used by another node a collision is detected when the TA wants to commit the modified data. Plurix solves this collision with a first wins strategy [2]. The first TA committing is always allowed to propagate the modified object into the DSM. A multicast commit message is sent to let the other nodes automatically recognise that they have now invalid data. Affected TAs are aborted by the operating system and immediately restarted with invalidated and then updated data. The TA itself and its program does not notice that it was aborted and restarted.

How much processing is done by a TA is a decision of the programmer. But because of the optimistic synchronisation scheme short TAs are preferred in the Plurix system.

When creating new long running applications the individual tasks are easily split up into transactions by identifying naturally arising sub-tasks such as input, processing and output.

Because all nodes always see the newest version of data within the DSM sequential consistency is implemented with the concept of isolated transactions – the transactional consistency – making a transactional operating system an ideal platform for an interactive virtual reality environment [3].

3. RELATED WORK

Traditionally, 3D online worlds are implemented as client-server based scenarios. In common network games the scene graph used mixes the global shared world information with local viewing data of the client to increase performance of the application and render process (e.g. Java3D). Typically, the server updates the global scene graph interpreting action messages from the clients and returning update information to them (figure 1). The server must compute the results of the received client actions, administrate the update process and send synchronisation messages. Thus the server tends to become a bottleneck both with respect to network load and to computational power. The clients themselves normally keep copies of the complete scene graph in their memory but again mix it with their local view to speed up rendering.

To reduce network load and relieve the server the clients usually have to calculate the behaviour of softbots and the motion of animated objects by themselves resulting in higher system requirements. Additionally, some strategies like dead reckoning [4] are necessary to correct data, for instance the

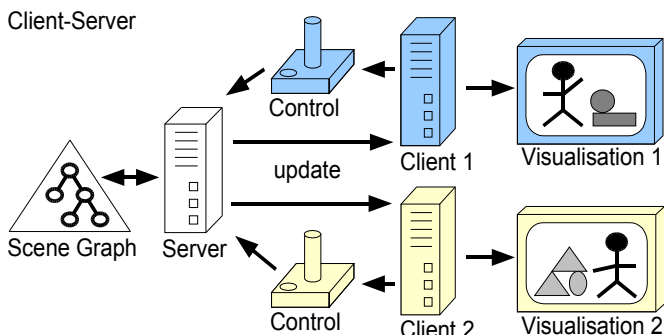


Figure 1: The traditional integration of a scene graph in a client-server architecture

position of a softbot, if the synchronisation message from the server differs from the interpolated data.

For a scene graph there are already many formats developed for and used by single station systems, for instance VRML, but these all lack the possibility to be simply expanded into a distributed design.

Explicit distributed approaches for a scene graph like Avocado [5], DIVE [6], Distributed Open Inventor [7] or Repo-3D [8] are either replicating the scene graph on each node according to some relaxed consistency models or use explicit update messages to achieve synchronisation. Therefore they are not fully suited for a transactional OS.

4. SCENE GRAPH

Every virtual reality environment needs some kind of underlying data structure to represent the design of the whole virtual world. In Plurix we designed our own scene graph but retain similarities to a distributed VRML scene.

To be easy shareable a scene graph should only contain the globally relevant parts of the virtual world and no mixing with local viewing and camera information from a single node may occur even if some nodes have replicated parts of the scene graph in their memory (figure 2). If a node modifies an object in the scene all other participating nodes will immediately see the changes because of the transactional consistency.

Therefore a node merely needs to calculate the local view of the avatar associated with this node and perhaps update the position of the avatar during movement or action. The position of other avatars and softbots in the virtual world can easily be accessed when needed and is always up to date so there is no need of explicit synchronisation.

Another benefit of a shared scene graph is the possible distribution of additional calculation to the cluster nodes. The behaviour of a softbot for instance is not calculated on every single participating node but on only one node. Nevertheless, all other nodes can see the result of the calculation at once. So it is possible that a node which creates an animated object or softbot, for example a little dog walking with his owner, can do the computation for this object on his machine and every other

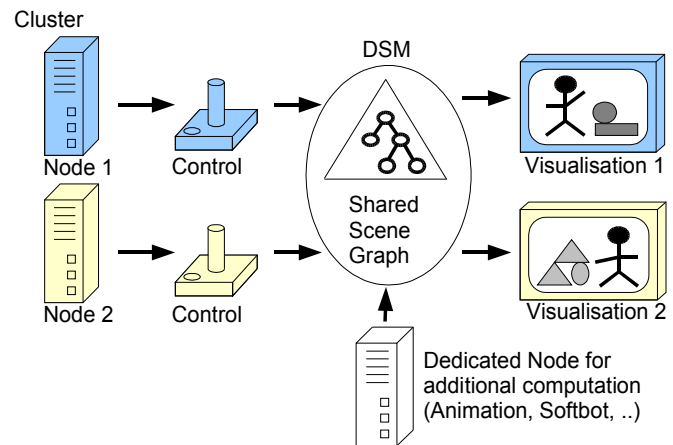


Figure 2: The use of a shared scene graph in a distributed operating system

avatar in view can also see the accompanying dog without having to calculate the positions and movements by themselves.

The corresponding TA doing this calculation is normally carried out on the node which has created the softbot object but this task may also be transferred to some possibly faster node or even a node which is not otherwise involved in the virtual world if it has enough resources to share. Thus a kind of dedicated compute server can easily be included and less powerful stations could also participate in a virtual reality environment.

The DSM scheme can elegantly redirect input and output from and to different nodes. A node will connect to its preferred avatar and can obtain control of it if the avatar is not already controlled by another node. Typically it will then also assume the perspective of the avatar. The associated render process connects to an avatar and is now able to see its view even if he is controlled by an other node. This enables presentation modes as well as observing the behaviour of a softbot.

5. RENDER PROCESS

To let an user explore and act inside the virtual world its graphic must be rendered. The standard render process in a traditional approach gathers all information needed for the next frame, calculates changed data such as object positions and sends the new frame to the graphic hardware for rendering. The procedure is repeated for each frame. All these steps together can be quite time consuming so with other computations done in a system the goal remains to achieve frame rates of at least 25 frames per second (FPS). In a client-server based system after receiving the required update information from the server a client can render the next frame without the risk of getting disturbed.

If this render process is transferred unchanged to a transactional operating system there is a high chance of a collision during rendering. Because of the comparatively long execution time of the render process the probability is high that in the meantime another node has updated some shared object data, for instance if an avatar in view is moved by another node. In this example the render process on the first node wants to visualise the position of the second avatar when its position gets updated. Now the first node has worked with the already gathered data and is preparing to send the results to the graphic adapter when the modifications of the second node causes an abort and restart of the render process. Since the data for the rendering process is already gathered early in the visualisation process the probability that another node modifies some of this data before the rendering has finished is substantial. The visible result is that the new frame is delayed and the last frame is still displayed. In the worst case a render process can constantly be slowed down so that no fluent visualisation is possible any more.

An attempt to stop other TAs from writing data during a render process is of marginal use because this will serialise the access to the virtual world for all nodes and leads to a traditional client-server treatment of the scene graph discarding

all advantages of a distributed system.

5.1 Multistage approach

An effective way to avoid collisions and slowdown of the render process is to split it up into at least two phases. The first stage, called the update TA, is the critical update process to gather the needed information for visualisation where a certain probability of transaction aborts is given. Therefore this TA has to be very short. It accesses the shared scene graph and stores the render information locally. Now the gathered data is separated from the shared global data and modifications in the scene graph can not affect the local data used to display the new frame afterwards. In order not to revert to a traditional network game scenario in which each client has a complete copy of the whole scene graph in its local memory, only selected data elements are copied. This data elements might be position coordinates or transformation matrices for visible objects. The much larger volume of geometry data of an object itself is likely to remain unchanged, so the probability of a collision associated with this kind of data is very low. Only references to the geometry of an object are stored locally and during rendering this data can be accessed normally from the DSM. If in spite of this the geometry of an object is changed, there is only one single abort of the render process; the next frame already contains the actualised geometry data and the frame rate instantly reverts to the one before.

The subsequent stage now consumes the local viewing data and carries out the remaining calculation for the render process without running the risk of being aborted - because the local data is not shared and no other node can access and invalidate it. If in the meantime some position data has been modified in the world after the update TA but before the finishing of this render TA, the local node will display the current frame with out-dated data. This will last one single frame at most but it is ensured that the render TA can finish. Already for the next frame the update TA will obtain fresh data and therefore will be up to date again.

With this multistage strategy an efficient render process can be realised. The risk of collisions during the update TA is still given, especially if many nodes are participating and modifying data. But compared to the render TA the execution time of the update TA is very short and the probability of collisions is considerably reduced.

6. PERFORMANCE MEASUREMENT

We have created an interactive 3D virtual reality application within the Plurix OS using the described concept of the shared scene graph and the multistage render process. Figure 3 shows the average frame rates of our prototype with an increasing number of participating nodes. The visualised world scene consists of 50.000 vertices plus one avatar for each node with additional 3.500 vertices. The PC configuration consists of all AMD Athlon processors running with 2.400+ MHz and an ATI Radeon 7500 PCI graphics adapter.

To demonstrate the performance of the virtual world in

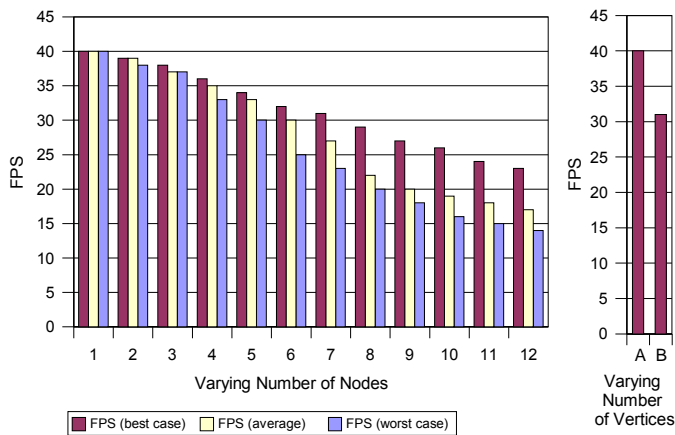


Figure 3 & 4: Frame rates of our interactive 3D world prototype

different situations three scenarios were tested. In the best case scenario no avatar movement or other modifications to the scene graph were made during measurement. The worst case scenario includes continuous movement for each avatar to generate permanent modifications within the scene graph, i.e. one modification per node per frame. The third case was an average scenario with a modification rate of about one third of the worst case, i.e. one modification per node per three frames.

It is visible that the gap between the best and the worst case scenario is not far apart. Thus the used transactional OS appears to manage adequately the load of the distributed accesses on the shared scene graph and resolving the occurred collisions.

A considerable part of the general decrease of the frame rate in each case results from the also increasing amount of vertices to be rendered with every additional avatar. This means even in the best case scenario with 12 nodes the render TA has almost twice as many vertices to render as with only one avatar. Figure 4 illustrates this issue. In case 'A' a single node is calculating a scene like in the best case scenario before with one avatar. Case 'B' shows the FPS of the same node now rendering a world with as many vertices as the equivalent amount in the best case scenario with 12 avatars. So it is recognisable that the frame rate also depends heavily on the amount of rendered data and thus the distribution of data is not the only reason for the decline of the frame rates.

It should also be mentioned that no additional software optimisations have been included within our prototype yet to speed up the rendering process like view frustum culling [9]. This is currently under construction and will substantially improve the frame rates.

7. CONCLUSIONS

Shared memory and transactional consistency offer a viable and elegant platform for virtual reality scenarios. An efficient implementation of an interactive 3D world has been built using a shared scene graph and a multistage render process.

The option to use dedicated stations to calculate additional tasks of a virtual reality application enables even very slow nodes in a very heterogenous PC cluster to work with adequate performance. The minimum system requirement is determined by the need to execute the render process only. Since the application runs in a peer-to-peer scenario reduced administration and server cost is achieved.

Clearly, a transactional operating system is a good platform for a distributed virtual reality environment with different participating nodes and is also offering an alternative to the traditional client-server based approach.

References

- [1] The PLURIX Project, www.plurix.de
- [2] M. Wende, M. Schoettner, R. Goeckelmann, T. Bindhammer, P. Schulthess, "Optimistic Synchronization and Transactional Consistency", in: Proceedings of the 4th International Workshop on Software Distributed Shared Memory, Berlin, Germany, 2002
- [3] M. Schoettner, M. Wende, R. Goeckelmann, U. Schmid, P. Schulthess, "A Gaming Framework for a Transactional DSM System", in: Proceedings of the Workshop on Distributed Shared Memory on Clusters, IEEE International Symposium on Cluster Computing and the Grid, Tokyo, Japan, 2003
- [4] L. Pantel, L. C. Wolf, "On the suitability of dead reckoning schemes for games", in: Proceedings of the 1st workshop on Network and system support for games, Braunschweig, Germany, 2002
- [5] H. Tramberend, "Avocado: A Distributed Virtual Environment Framework", Dissertation, Universität Bielefeld, Technische Fakultät, 2003
- [6] C. Carlsson, O. Hagsand, "DIVE - A Platform for Multi-user Virtual Environments", *Computer and Graphics*, 17(6), pp. 663-669, 1996
- [7] G. Hesina, D. Schmalstieg, A. Fuhrmann, W. Purgathofer, "Distributed Open Inventor: A Practical Approach to Distributed 3D Graphics", in: Proceedings of the ACM Symposium on Virtual Reality Software and Technology, Vienna University of Technology, 1999
- [8] B. MacIntyre, S. Feiner, "Repo-3D - A Distributed 3D Graphics Library", in: Proceedings of the ACM SIGGRAPH '98, Orlando, Florida, 1998
- [9] U. Assarsson, T. Moeller, "Optimized View Frustum Culling Algorithms", Technical Report 99-3, Chalmers University of Technology, 1999